Techniques for Holistic Quantification of Performance, Portability, and Productivity

Jason Sewall, John Pennycook, Doug Jacobsen
Intel Corporation
P3HPC Forum 2020

Acknowledgements: University of Bristol

Intel, the Intel logo, Intel® Xeon Phi™, Intel® Xeon® Processor are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others. See Trademarks on intel.com for full list of Intel trademarks.

Ψ, Quantitative Performance Portability

The **harmonic mean** of performance efficiency on a set of platforms

$$\Psi(a,p,H) = \begin{cases} \dfrac{|H|}{\sum_{i \in H} \dfrac{1}{e_i(a,p)}} & \text{if } i \text{ is supported } \forall i \in H \\ 0 & \text{otherwise} \end{cases}$$

$e_i(a,p)$ = efficiency of application a for input problem p.
H = set of platforms of interest

- Has desirable properties:
  - Supports many definitions of application/problem/efficiency/platform
  - Improving any result improves the metric
  - Non-portable code (i.e. performance of 0 on a platform) has score of 0
  - Easy to compute

S.J. Pennycook, J.D Sewall, V.W. Lee, "Implications of a Metric for Performance Portability" in *Future Generation Computer Systems*, 2017

© 2020 Intel Corporation

Code Divergence: quantitative portability/productivity

- Previous metric is completely orthogonal to productivity
  - 'Duct-tape' apps with totally distinct code for each platform compared to single-source applications
- We introduced code divergence:
  - Average of code distance between each pair of platforms
  - Use Jaccard distance of lines of code: $\frac{|c_i \cup c_j| - |c_i \cap c_j|}{|c_i \cup c_j|}$
- Has desirable properties:
  - 'Duct-tape' apps have divergence of 1
  - Single-source apps have divergence of 0
  - Partially specialized codes fit in between

[1] S.L Harrell, J. Kitson et al., "Effective Performance Portability", in *Proceedings of P3HPC*, 2018

© 2020 Intel Corporation

Code Divergence – Simple Example

Codebase 1

shared.cpp:

```
void whereami()
{
#ifdef CPU
  printf(“CPU\n”); ← 1 line specific to CPU
#else
  printf(“GPU\n”); ← 1 line specific to GPU
#endif
}
```

Divergence = (8 – 6) / 8 = 0.25

Codebase 2

cpu.cpp: ← entire file specific to CPU

```
void whereami()
{
  printf(“CPU\n”);
}
```

gpu.cpp: ← entire file specific to GPU

```
void whereami()
{
  printf(“GPU\n”);
}
```

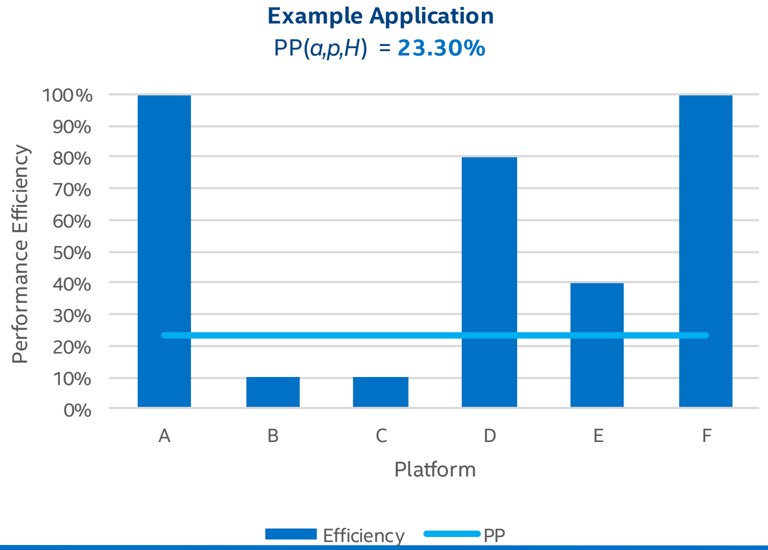
Divergence = ((4 + 4) – 0) / 8 = 1.0

© 2020 Intel Corporation

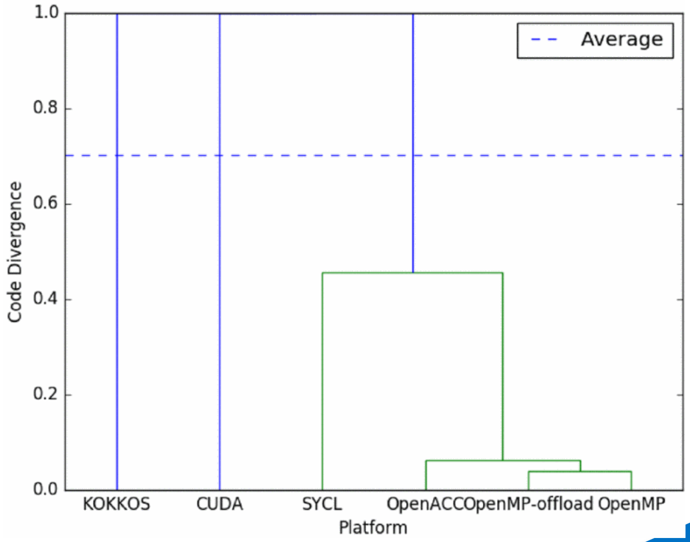
Combining metrics

- These metrics are complementary: one captures performance & portability, another productivity

Performance & Portability



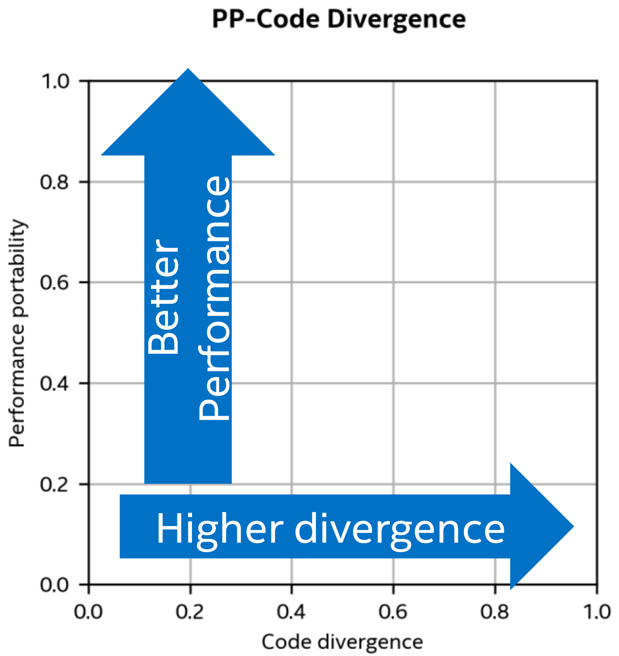
Productivity



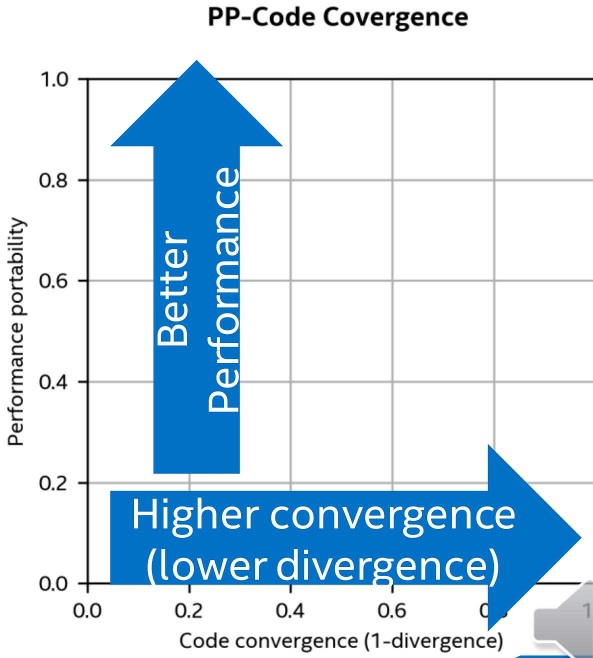
© 2020 Intel Corporation

Combining metrics

- Plotting metrics in 2D can be illuminating. Select platform set, use for both PP and Code Divergence computation

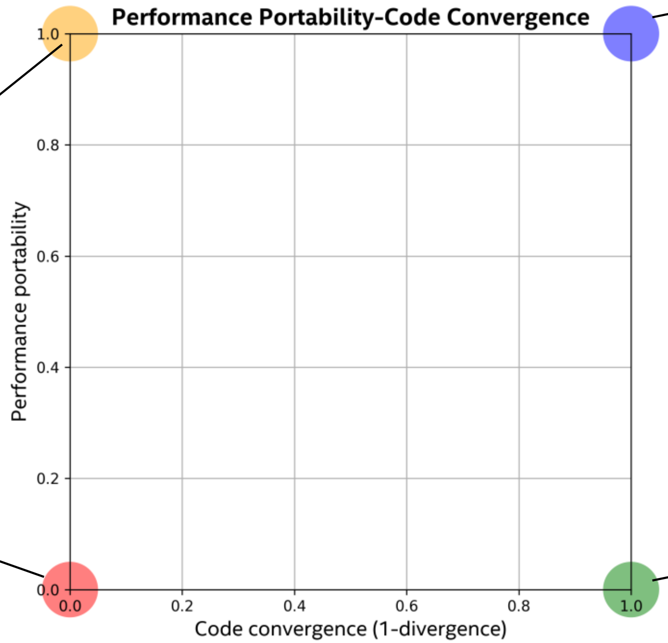


- Code Divergence is the right metric, but it doesn't plot well
- Rather, use 'Code Convergence' (1- Code Divergence)



© 2020 Intel Corporation

Boxing the combined graph



Perfect performance portability, total divergence

Perfect performance portability, total convergence

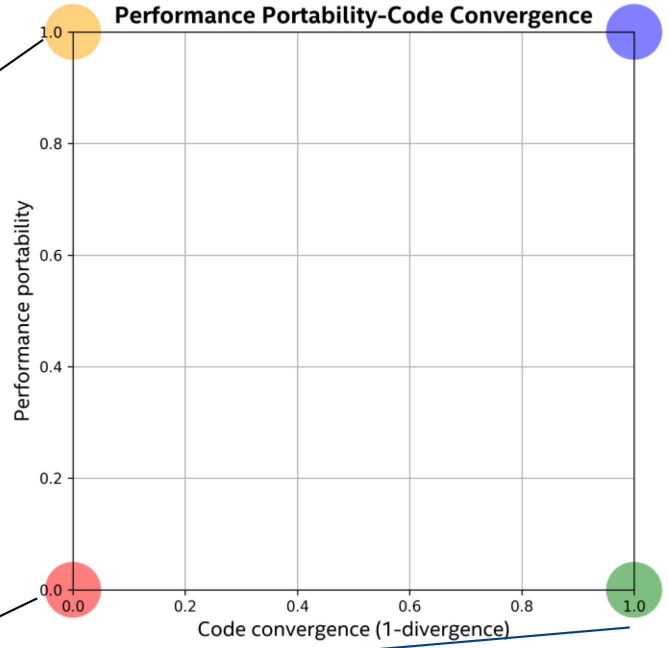
No performance portability, total divergence

No performance portability, total convergence

© 2020 Intel Corporation

Boxing the combined graph

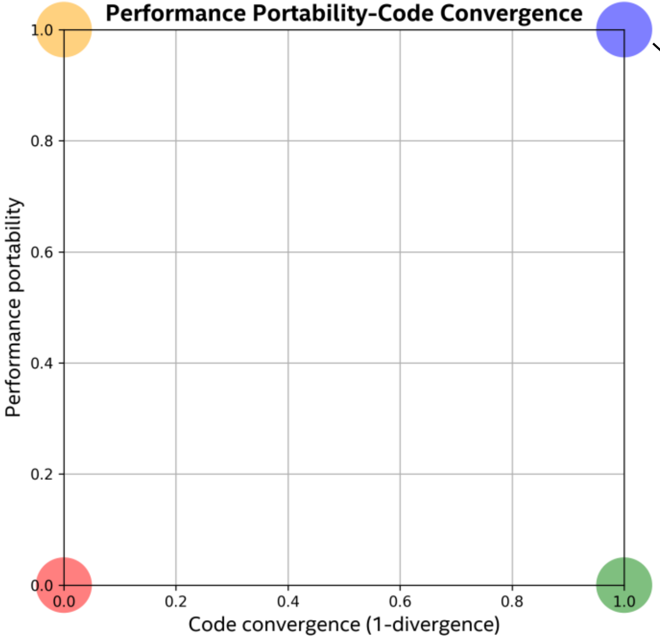
Can always construct this by combining (disparate) implementations that are best for each platform of interest.
AKA "the duct tape solution"



PP=0 axis is of little interest---code doesn't run on one or more platforms!

© 2020 Intel Corporation

Boxing the combined graph

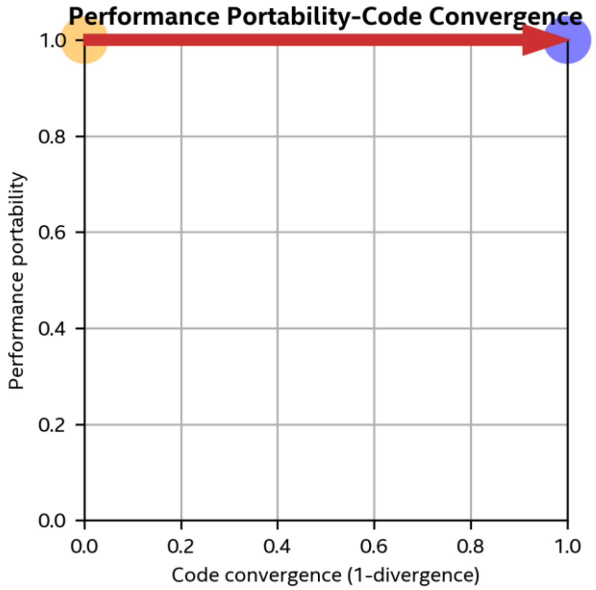


The Promised Land: single-source, best performance on all platforms (not realistic!)

© 2020 Intel Corporation

Translation in the PP-CC Plane

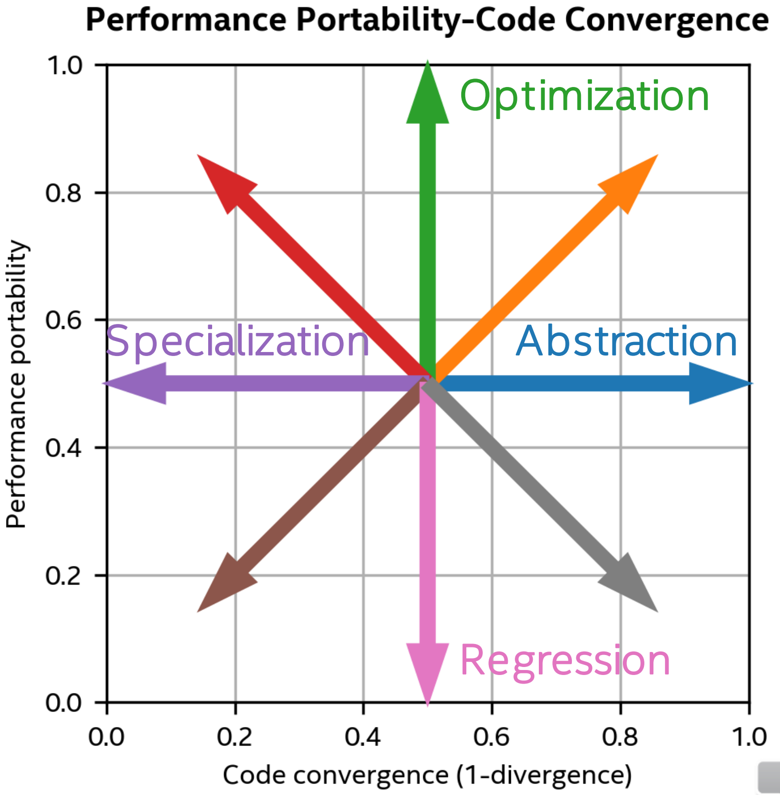
- Given 'duct-tape' application, how do we get the best case?
- PP and CC are linked
- CC = 1/PP = 1 are very hard to achieve, but settling for close to these limits has opportunities
  - Heroic (magic?) compilers needed for strict increases



© 2020 Intel Corporation

Translation in the PP-CC Plane

- Movement can be broadly described in terms of the axes (assume fixed platform set)
  - More abstraction -> Higher convergence
  - Less abstraction == specialization -> lower convergence
  - Better optimized code -> Higher PP
  - Regressed code -> Lower PP
    – Never a standalone goal, but a side-effect

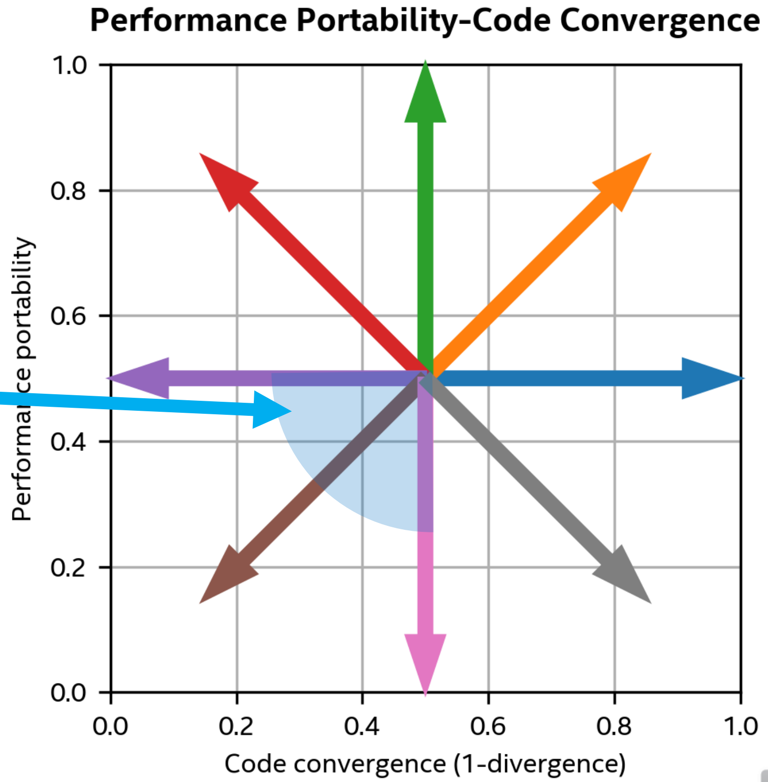


© 2020 Intel Corporation

Translation in the PP-CC Plane

Specialization/Regression quadrant:
Includes Specialization/Regression axes
Almost never an actual goal.
Hypothetical causes:

- Feature added that decreases performance (possibly specialized)

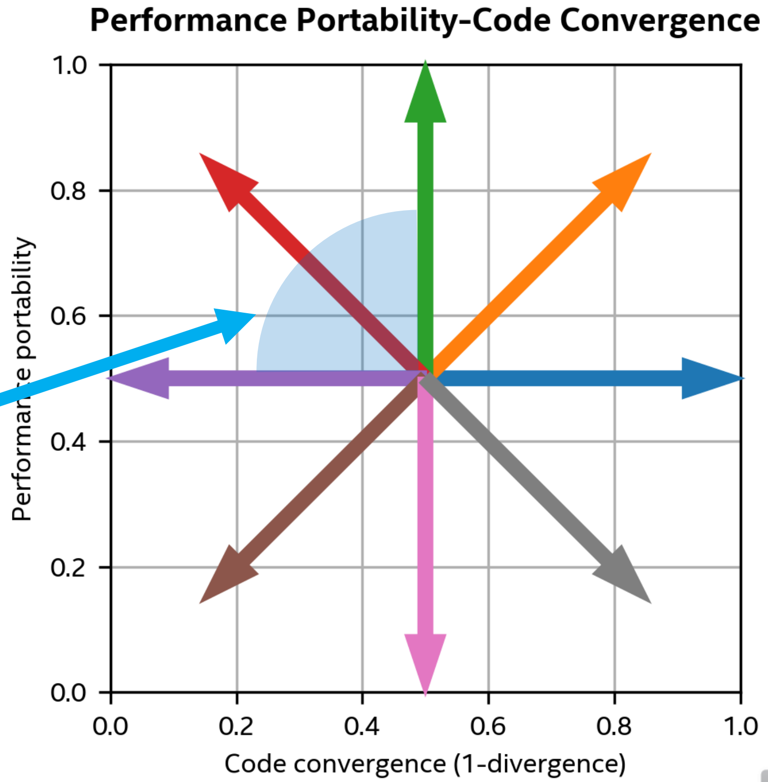


© 2020 Intel Corporation

Translation in the PP-CC Plane

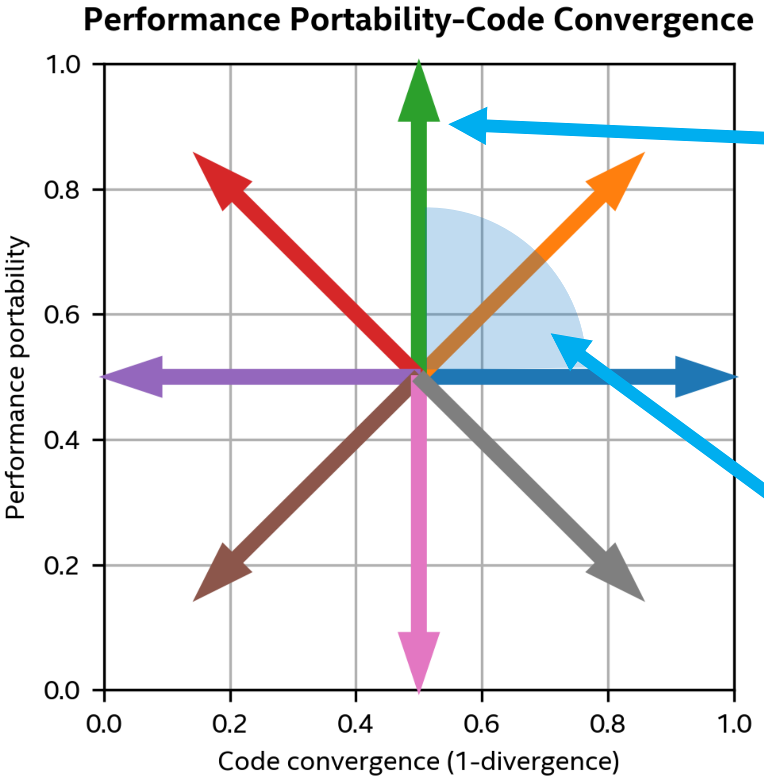
Specialization/Optimization quadrant:
Very typical scenario:
1. Relatively convergent codebase
2. Identify lagging platforms
3. Introduce specializations to improve performance

Considerations:
- How fine-grained is specialization mechanism?
- What is the trade-off of CC vs. PP?



© 2020 Intel Corporation

Translation in the PP-CC Plane



Optimization axis:

A change to common codebase increases overall PP

- Due to performance gains on a single platform or across multiples

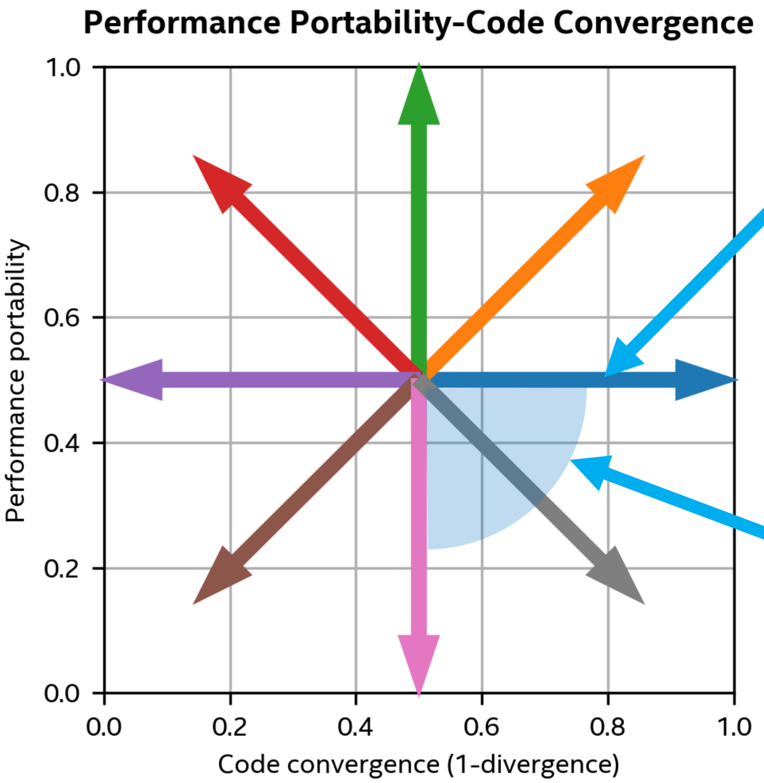
Optimization/Abstraction quadrant:

Code is more convergent and more performant!

- Tool/backend improvements
- Magic compilers

© 2020 Intel Corporation

Translation in the PP-CC Plane



Abstraction axis:

Refactoring code increases CC, doesn't affect performance.

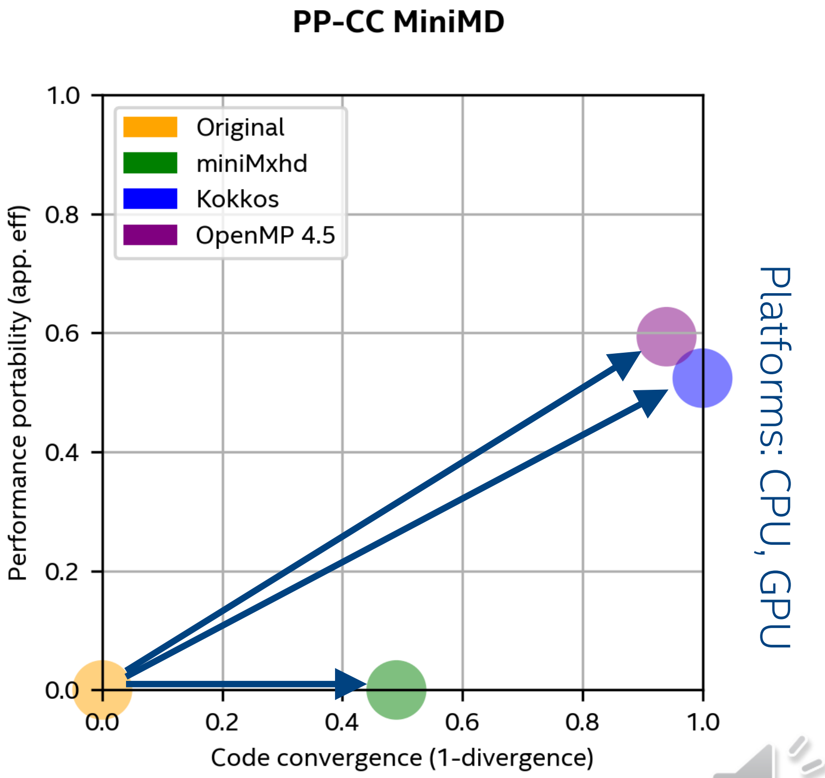
Regression/Abstraction quadrant:

Abstraction penalties arise.
As abstractions are introduced, performance on some platforms may decrease

© 2020 Intel Corporation

MiniMD

- Venerable Mantevo proxy for LAMMPS
- Original OpenMP code is CPU-only
- miniMxhd developed by Intel
  - Highest performing CPU version
- Kokkos version developed by Sandia
  - Highest-performing GPU version
- OpenMP 4.5 version developed by Intel
  - Achieves highest PP due to highest CPU performance among GPU-supporting codes



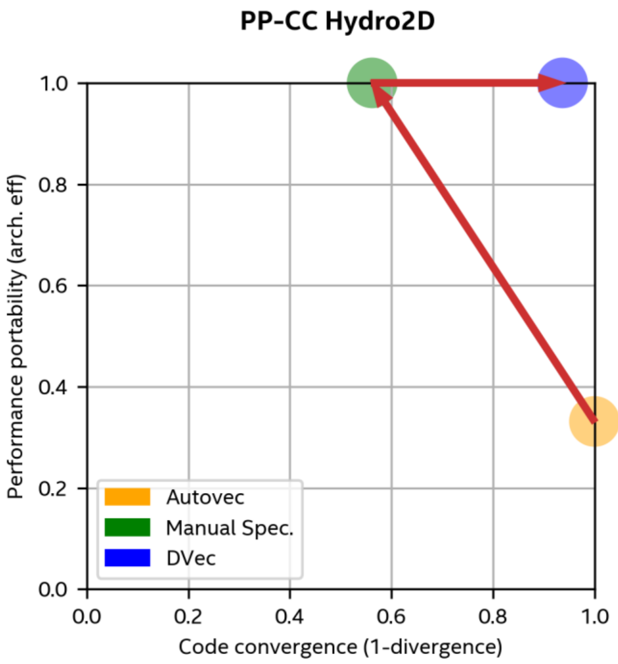
Pennycook, Sewall, Hammond “Evaluating the Impact of Proposed OpenMP 5.0 Features on Performance, Portability, and Productivity”. SC P3HPC 2018

https://github.com/Mantevo/miniMD

© 2020 Intel Corporation

Hydro2D

- CEA benchmark code
- OpenMP code optimized by Intel
- Autovectorization performance poor, but single codebase
- Manual vectorization using intrinsics introduced for each SIMD ISA
  - High performance, less convergence
- Refactoring to a common vector abstraction results in matching performance, higher convergence.



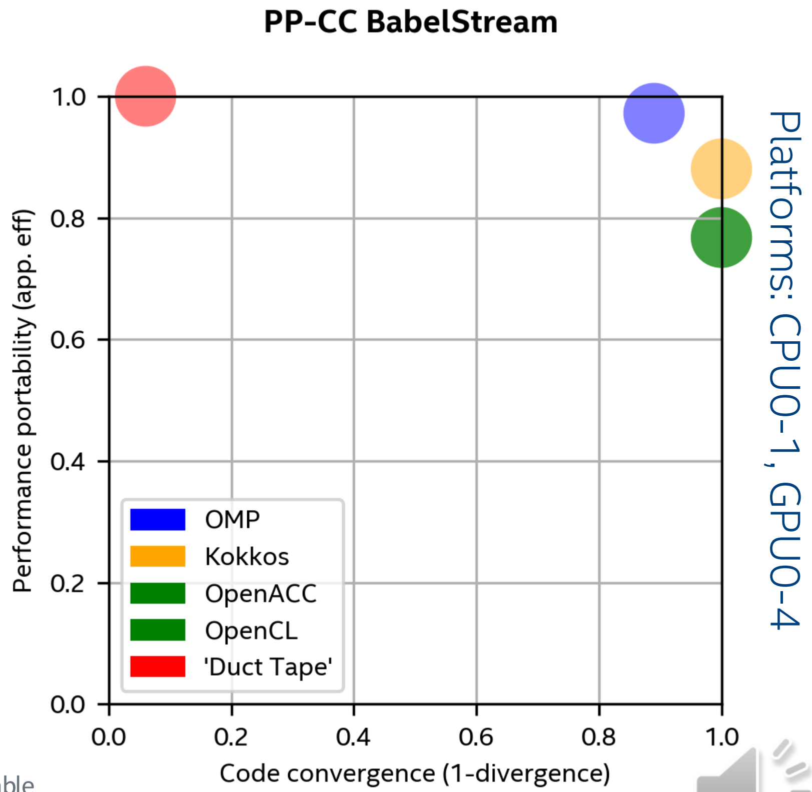
Platforms:
CPU0,CPU1,CPU2,CPU3

Sewall, Colin de Verdiere, "From 'Correct' to 'Correct and Efficient': A Case Study with Hydro2D". Intel Xeon Phi Programming Gems, Reinders et al. eds 2015.

© 2020 Intel Corporation

Babelstream

- Multi-platform STREAM code
- Many implementations
  - Minimal effort made to reduce divergence
- Synthetic 'Duct Tape' app constructed from all codes
  - Highest PP, low CC
- OpenMP has slightly different codes for CPU and GPU (omp target)
  - High PP, some reduced CC
- Kokkos, OpenACC, and OpenCL CC > 0.999
  - OpenACC and OpenCL have nearly identical PP



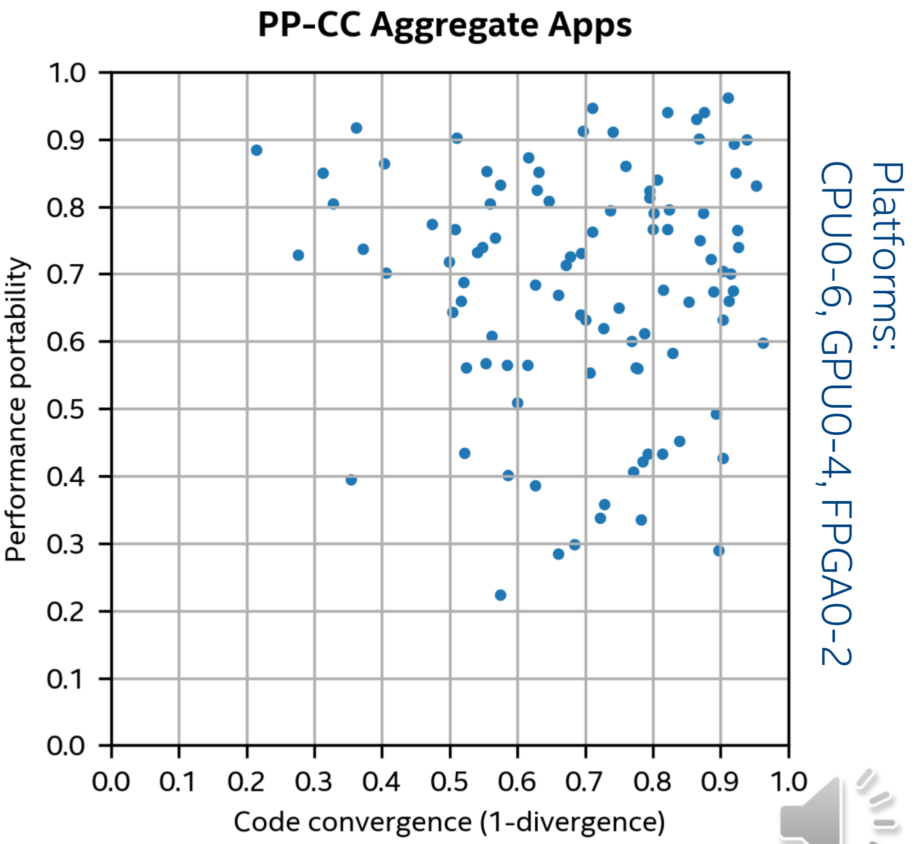
Deakin T, Price J, Martineau M, McIntosh-Smith S. GPU-STREAM v2.0: Benchmarking the achievable memory bandwidth of many-core processors across diverse parallel programming models. 2016. Paper presented at P^3MA Workshop at ISC High Performance, Frankfurt, Germany.

https://github.com/UoB-HPC/BabelStream

© 2020 Intel Corporation

Aggregate analysis

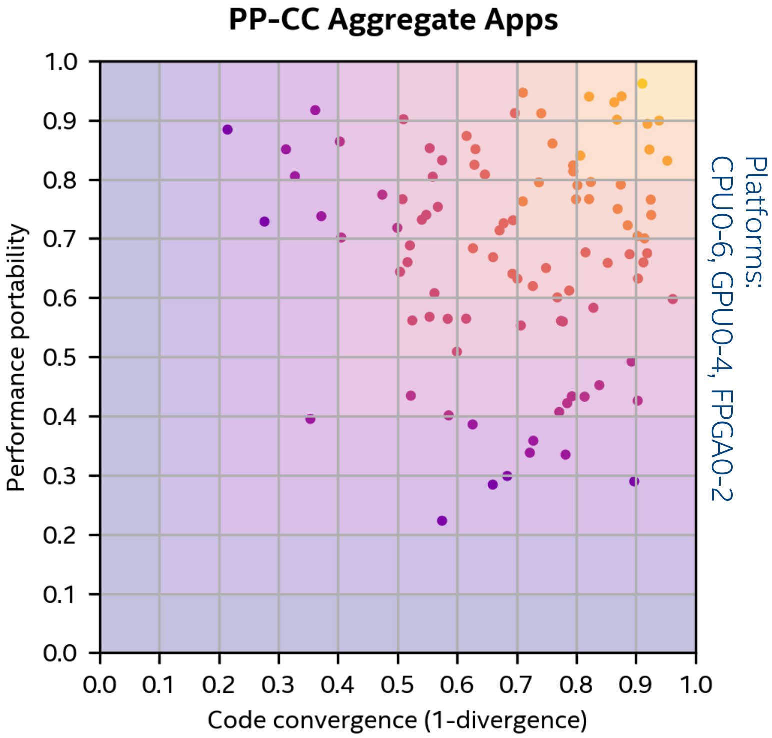
- How can we apply this to a group of codes running on the same platforms?
  - What does it tell us?
  - How can we use it to measure success
    - Of codes
    - Of tools
- Synthetic data presented for exposition

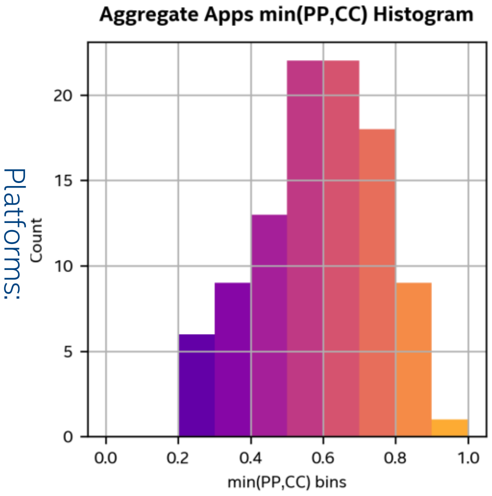


© 2020 Intel Corporation

Aggregate analysis

- Given goal of having codes in the upper-right part of chart:
  - Score each point with min(PP,CC)
  - Sort into bins of length 1/10
- Extract statistics
  - Minimum is 0.21
  - Median is 0.60
  - Maximum is 0.91

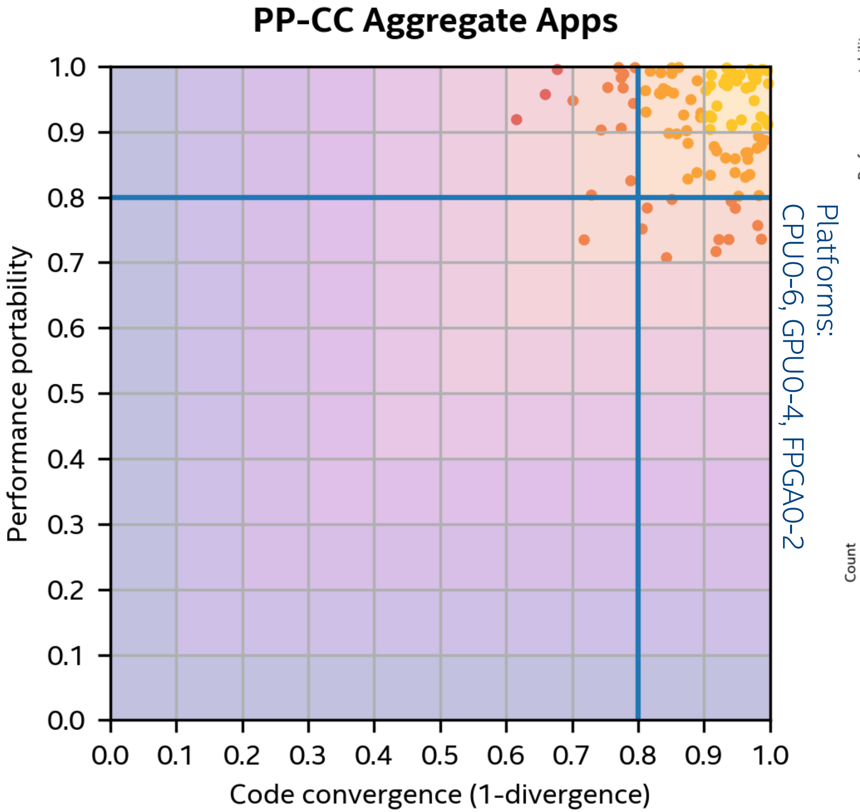


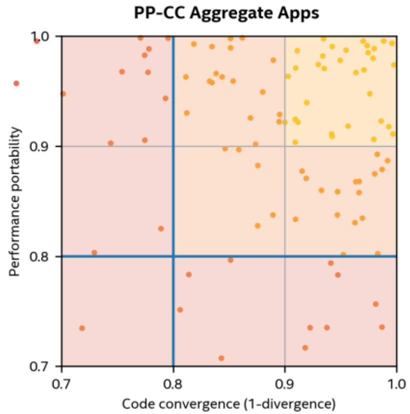


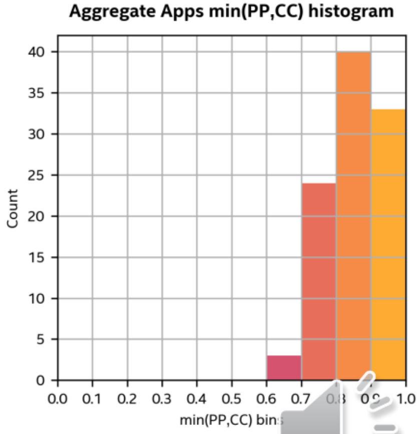
© 2020 Intel Corporation

Aggregate analysis

- Optimized codes/tools
- Set goal that 50% of codes must score 0.8 or higher
- Stats:
  - Minimum is 0.62
  - Median is 0.86
  - Maximum is 0.993







© 2020 Intel Corporation

Review

- Performance portability and code divergence scores are complementary in a P3 context
- Combining them in a 2D plot—with attention to platforms and adjustment to code convergence—can reveal useful trends and patterns, both with single applications and aggregates
  - Seeing current status and setting goals in the PP-CC plane can give clues on how to proceed to improve application
- Methods derived from this are used as internal scoring metrics inside Intel to guide us to a more P3 future

© 2020 Intel Corporation

Future work

- The usefulness of P3 is inherently subjective
  - Need more data on what people are satisfied with, and how it looks in these visualization
- Varying platforms in these graphs is a promising direction for understanding sensitivity in toolchains and hardware
- "Language shootout" applications for understanding relative strengths/weaknesses among programming tools
- More real data sets analyzed and published!

© 2020 Intel Corporation

Legal Disclaimers

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Performance results are based on testing as of the publication date of the referenced papers and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure. Configurations:
Slide 16 – Measured by Intel; J. Pennycook, J. Sewall, J. Hammond "Evaluating the Impact of Proposed OpenMP 5.0 Features on Performance, Portability, and Productivity". SC P3HPC 2018
Slide 17 – Measured by Intel & CEA: J. Sewall, G. Colin de Verdiere, "From 'Correct' to 'Correct and Efficient': A Case Study with Hydro2D". Intel Xeon Phi Programming Gems, Reinders et al. eds 2015.
Slide 18 – Measured by University of Bristol: T. Deakin, J. Price, M. Martineau, S. McIntosh-Smith. GPU-STREAM v2.0: Benchmarking the achievable memory bandwidth of many-core processors across diverse parallel programming models. 2016. Paper presented at P^3MA Workshop at ISC High Performance, Frankfurt, Germany.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804

Intel, the Intel logo, Xeon, and Xeon Phi are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

© 2020 Intel Corporation

intel®